Error Bounds from Extra Precise Iterative Refinement[*]

James Demmel[†] Yozo Hida[‡] W. Kahan[§] Xiaoye S. Li[¶] Soni Mukherjee[‖]
E. Jason Riedy[**]

March 19, 2005

Abstract

We present the design and testing of an algorithm for iterative refinement of the solution of linear equations, where the residual is computed with extra precision. This algorithm was originally proposed in the 1950s [22] and analyzed in the 1960s [6, 18] as a means to compute very accurate solutions to all but the most ill-conditioned linear systems. However two obstacles have until now prevented its adoption in standard subroutine libraries like LAPACK: (1) There was no standard way to access the higher precision arithmetic needed to compute residuals, and (2) it was unclear how to compute a reliable error bound for the computed solution. The completion of the new BLAS Technical Forum Standard [5] has recently removed the first obstacle. To overcome the second obstacle, we show how a single application of iterative refinement can be used to compute an error bound in any norm at small cost, and use this to compute both an error bound in the usual infinity norm, and a componentwise relative error bound.

We report extensive test results on over 6.2 million matrices of dimension 5, 10, 100, and 1000. As long as a normwise (resp. componentwise) condition number computed by the algorithm is less than $1/\max\{10,\sqrt{n}\}\varepsilon_w$, the computed normwise (resp. componentwise) error bound is at most $2\max\{10,\sqrt{n}\}\cdot\varepsilon_w$, and indeed bounds the true error. Here, n is the matrix dimension and $\varepsilon_w = 2^{-24}$ is the roundoff error. Residuals were computed in double precision (53 bits of precision). In other words, the algorithm can guarantee a tiny error at neglible extra cost for most linear systems.

For worse conditioned problems, we also get small correct error bounds in over 89% of cases.

[*]Computer Science Division Technical Report UCB//CSD-04-1344, University of California, Berkeley, 94720. This research was supported in part by the NSF Cooperative Agreement No. ACI-9619020; NSF Grant Nos. ACI-9813362 and CCF-0444486; the DOE Grant Nos. DE-FG03-94ER25219, DE-FC03-98ER25351, and DE-FC02-01ER25478; and the National Science Foundation Graduate Research Fellowship. The authors wish to acknowledge the contribution from Intel Corporation, Hewlett-Packard Corporation, IBM Corporation, and the National Science Foundation grant EIA-0303575 in making hardware and software available for the CITRIS Cluster which was used in producing these research results.

[†]Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720 (`demmel@cs.berkeley.edu`).

[‡]Computer Science Division, University of California, Berkeley, CA 94720 (`yozo@cs.berkeley.edu`).

[§]Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720 (`wkahan@cs.berkeley.edu`).

[¶]Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (`xsli@lbl.gov`).

[‖]Computer Science Division and Mathematics Dept., University of California, Berkeley, CA 94720.

[**]Computer Science Division, University of California, Berkeley, CA 94720 (`ejr@cs.berkeley.edu`).

1 Introduction

Iterative refinement is a technique for improving the accuracy of the solution of a system of linear equations $Ax = b$. Given some basic solution method (such as Gaussian Elimination with Partial Pivoting – GEPP), the basic algorithm is as follows:

```
Input: An $n \times n$ matrix $A$, and an $n \times 1$ vector $b$
Output: A solution vector $x^{(i)}$ approximating $x$ in $Ax = b$, and
        an error bound $\approx \|x^{(i)} - x\|_\infty / \|x\|_\infty$
Solve $Ax^{(1)} = b$ using the basic solution method
$i = 1$
repeat
    Compute residual $r^{(i)} = Ax^{(i)} - b$
    Solve $A\,dx^{(i+1)} = r^{(i)}$ using the basic solution method
    Update $x^{(i+1)} = x^{(i)} - dx^{(i+1)}$
    $i = i + 1$
until $x^{(i)}$ is "accurate enough"
return $x^{(i)}$ and an error bound
```

Algorithm 1: Basic iterative refinement

(Note that $x^{(i)}$ is a vector, and we use the notation $x_j^{(i)}$ to mean the j-th component of $x^{(i)}$.)

This can be thought of as Newton's method applied to the linear system $f(x) = Ax - b$. In the absence of error, Newton's method should converge immediately on a linear system, but the presence of rounding error in the inner loop of the algorithm prevents this immediate convergence and makes the behavior and analysis interesting.

The behavior of the algorithm depends strongly on the accuracy with which the residual $r^{(i)}$ is computed. We use *working precision* ε_w to denote the precision with which all input variables are stored. The basic solution method is used to solve $Ax = b$ and $A\,dx = r$ in working precision. In our numerical experiments, working precision is IEEE754 single precision, *i.e.* $\varepsilon_w = 2^{-24}$. This idea was first proposed by Snyder in 1955 [22]. Analyses of Wilkinson [6] and Moler [18] show that if the residual is computed to about double the working precision then as long as the condition number of A is not too large (sufficiently less than $1/\varepsilon_w$) the solution $x^{(i)}$ will converge to roughly working precision. Furthermore, the cost of this high accuracy is negligible: $O(n^2)$ on top of the $O(n^3)$ cost of straightforward Gaussian elimination.

The omission so far of this attractive algorithm from standard libraries like LAPACK [1] (or its predecessor LINPACK [8]) can be attributed to two obstacles: (1) the lack of a portable implementation of higher precision arithmetic to compute the residual, and (2) a way to compute a truly reliable error bound. The recent completion of the BLAS Technical Forum Standard [5] has eliminated the first obstacle, and our goal in this paper is to eliminate the second obstacle.

Section 2 summarizes the error analysis of Algorithm 1 above (further details of this and other sections may be found in [7]). This analysis justifies our stopping criterion and a reliable bound for the *normwise relative error*

$$\frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty}. \tag{1}$$

Here and later $x = A^{-1}b$ denotes the exact solution, assuming A is not singular.

Second, we observe that the entire algorithm is *column scaling invariant.* More precisely, if we assume that (1) our basic solution scheme is GEPP without any Strassen-like implementation [24], (2) that no over/underflow occurs, and (3) C is any diagonal matrix whose diagonal entries are powers of the floating point radix β ($\beta = 2$ in the case of IEEE754 floating point standard arithmetic [2]), then replacing the matrix A by $A_c \equiv AC$ results in *exactly* the same roundoff errors being committed by Algorithm 1: The exact solution x_c of the scaled system $A_c x_c = b$ satisfies $x_c = C^{-1}x$ where $Ax = b$, and every intermediate floating point approximation $x_c^{(i)} = C^{-1}x^{(i)}$.

This means that a single application of Algorithm 1 (producing a sequence of approximations $x^{(i)}$) can be thought of as implicitly producing the sequence $x_c^{(i)}$ for the scaled system $A_c x_c = b$. This will mean that at a modest extra cost, we will be able to modify Algorithm 1 to compute the stopping criterion and error bound for x_c for any diagonal scaling C. (The extra cost is $O(n)$ per iteration, whereas one iteration costs $O(n^2)$ if A is a dense matrix.) In other words we will be able to cheaply compute a bound on the *scaled relative error*

$$\frac{\|C^{-1}(x^{(i)} - x)\|_\infty}{\|C^{-1}x\|_\infty} \tag{2}$$

for any scaling C.

Of the many C one might choose, a natural one would be $C \approx \mathrm{diag}(x_j)$, so that each component $x_{c,j} \approx 1$. This means that the scaled relative error (2) measures the componentwise relative error in the solution. There are two conditions for this to work. First, no component of x can equal 0, since in this case no finite componentwise relative error bound exists (unless the component is computed exactly). Second, the algorithm must converge (since C, which is computed on-the-fly, will affect the stopping criterion too).

Section 2 summarizes the precise stopping criterion and error bound for Algorithm 1. One outcome of this analysis are two condition numbers that predict the success of iterative refinement. Let n be the matrix dimension and ε_w be the working precision. Then if the normwise condition number $\kappa_{\mathrm{norm}}(A) < 1/\gamma\varepsilon_w$, where $\gamma = \max\{10, \sqrt{n}\}$, the error analysis predicts convergence to a small normwise error and error bound. Similarly, if the componentwise condition number $\kappa_{\mathrm{comp}}(A) < 1/\gamma\varepsilon_w$, the error analysis predicts convergence to small componentwise error and error bound. This is borne out by our numerical experiments described below.

Our ultimate algorithm, Algorithm 2, is described in Section 3. Algorithm 2 differs from Algorithm 1 in several important ways, beyond computing both normwise and componentwise error bounds. In particular, if consecutive increments $dx^{(i)}$ are not decreasing rapidly enough, the algorithm switches to representing $dx^{(i)}$ in *doubled working precision*, *i.e.* by a pair of working precision arrays representing (roughly) the leading and trailing bits of $dx^{(i)}$ as though it were in double precision. Iteration continues subject to the same progress monitoring scheme. This significantly improves accuracy on the most ill-conditioned problems.

Extensive numerical tests on over two million 100×100 test matrices are reported in Section 6. (Similar results were obtained on two million 5×5 matrices, two million 10×10 matrices, and $2 \cdot 10^5$ 1000×1000 matrices.) These test cases include a variety of scalings, condition numbers, and ratios of maximum to minimum components of the solution.

We summarize the results of these numerical tests. First we consider the normwise error and error bound. For not-too-ill-conditioned problems, those where $\kappa_{\mathrm{norm}}(A) < 1/\gamma\varepsilon_w$, Algorithm 2 *always* computed an error bound of at most $2\gamma\varepsilon_w$, which exceeded the true error. Since the algorithm computes $\kappa_{\mathrm{norm}}(A)$, these cases are easily recognized. For even more ill-conditioned problems, with normwise condition numbers κ_{norm} ranging up past ε_w^{-2}, Algorithm 2 still gets similarly small normwise error bounds and true errors in 96.4% of cases.

Next we consider the componentwise error and error bound. For not-too-ill-conditioned problems, those where $\kappa_{\mathrm{comp}}(A) < {}^1\!/\!\gamma\varepsilon_w$, Algorithm 2 *always* computed an error bound of at most $2\gamma\varepsilon_w$, which again exceeded the true error. The number of iterations required was at most 4, with a median of 2. Since the algorithm computes $\kappa_{\mathrm{comp}}(A)$, these cases are easily recognized. For even more ill-conditioned problems, with componentwise condition numbers κ_{comp} ranging up past ε_w^{-2}, Algorithm 2 still gets similarly small componentwise error bounds and true errors in 94% of cases.

Our use of extended precision is confined to two routines for computing the residual $r^{(i)} = Ax^{(i)} - b$, one where all the variables are stored in working precision, and one where $x^{(i)}$ is stored as a pair of vectors each in working precision: $r^{(i)} = Ax^{(i)} + Ax_t^{(i)} - b$. The first operation $r^{(i)} = Ax^{(i)} - b$ is part of the recently completed new BLAS standard [5], for which portable implementations exist [15]. The second operation $r^{(i)} = Ax^{(i)} + Ax_t^{(i)} - b$ was not part of the new BLAS standard because its importance was not recognized. Nevertheless, it is straightforward to implement in a portable way using the same techniques as in [15].

The rest of this paper is organized as follows. Section 2 summarizes the error analysis of our algorithm, including their invariance under column scaling; see [7, Sec. 2] for details. Section 3 describes the ultimate algorithm, Algorithm 2. Section 4 describes related work, Section 5 describes the testing configuration, including how test matrices are generated. Section 6 presents the results of numerical tests. Finally Section 7 draws conclusions and describes future work.

2 Error Analysis

Here we summarize an error analysis to justify our choices for termination criteria, error estimates, and the accuracy with which each step is performed. For details see [7, Sec.2]. Let the true forward error of iteration i be $e^{(i)} \stackrel{\mathrm{def}}{=} x^{(i)} - x$. Our main contributions are (1) to show that the termination criteria and error estimates apply not just to $\|e^{(i)}\|_\infty$ but to any diagonally scaled error $\|Ce^{(i)}\|_\infty$, and (2) to derive easily an estimated condition number (depending on C) that can be used to identify linear systems when convergence to an accurate solution is "guaranteed" as opposed to extremely ill-conditioned cases where such convergence is only high likely.

Our describe the three different precisions our algorithm uses ε_w, ε_x, and ε_r. In our experiments, working precision $\varepsilon_w = 2^{-24}$ is IEEE754 single precision [2]. Our analysis ignores over/underflow. The input data A and b are stored in working precision. The factorization of A is done and results stored using ε_w. The residual $r^{(i)}$ and step $dx^{(i)}$ are also stored in working precision, but the solution $x^{(i)}$ is stored and updated to precision $\varepsilon_x \le \varepsilon_w$, where possibly $\varepsilon_x \le \varepsilon_w^2$ if necessary for componentwise convergence. The criteria for choosing ε_x is discussed below. Residuals are calculated to extra precision ε_r with $\varepsilon_r \ll \varepsilon_w$ (typically $\varepsilon_r \le \varepsilon_w^2$). For our single-precision experiments, the residual is calculated in double precision with $\varepsilon_r = 2^{-53}$ and additional exponent range. The computed $x^{(i)}$ is carried either in single ($\varepsilon_x = \varepsilon_w = 2^{-24}$) or in a doubled single precision ($\varepsilon_x = \varepsilon_w^2 = 2^{-48}$).

Using this notation, the computed results $r^{(i)}$, $dx^{(i+1)}$, and $x^{(i+1)}$ from iteration i of Algorithm 1 satisfy the expressions

$$\begin{aligned}
r^{(i)} &= Ax^{(i)} - b + \delta r^{(i)} && \text{where} \quad |\delta r^{(i)}| \le n\varepsilon_r(|A|\cdot|x^{(i)}| + |b|) + \varepsilon_w|r^{(i)}|; && (3)\\
dx^{(i+1)} &= (A + \delta A^{(i+1)})^{-1} r^{(i)} && \text{where} \quad |\delta A^{(i+1)}| \le 3n\varepsilon_w|L|\cdot|U|; \text{ and} && (4)\\
x^{(i+1)} &= x^{(i)} - dx^{(i+1)} + \delta x^{(i+1)} && \text{where} \quad |\delta x^{(i+1)}| \le \varepsilon_x|x^{(i+1)}|. && (5)
\end{aligned}$$

Absolute values of matrices and vectors are interpreted elementwise.

Classical analyses in [6, 18] show that if we ignore the errors $\delta r^{(i)}$ and $\delta x^{(i+1)}$, then $\|dx^{(i+1)}\|_\infty$ decreases by a factor of at most $O(\varepsilon_w)\|A\|_\infty\|A^{-1}\|_\infty$ at every step. Therefore if $\kappa_\infty(A) \equiv \|A\|_\infty\|A^{-1}\|_\infty$ is sufficiently less than $1/\varepsilon_w$, the solution $x = x^{(1)} - \sum_{j=2}^{\infty} dx^{(j)}$ will converge like a geometric sum, with the norm of the error $e^{(i)} = \sum_{j=i+1}^{\infty} dx^{(j)}$ also decreasing geometrically.

If we use $\rho_{\max} = \max_{j\le i} \frac{\|dx^{(j+1)}\|_\infty}{\|dx^{(j)}\|_\infty}$ to estimate the rate of convergence, we may use the estimate $\|e^{(i)}\|_\infty \le \frac{\|dx^{(i+1)}\|_\infty}{(1-\rho_{\max})}$ as the basis of our error bound. This will be reliable as long as convergence is fast enough, i.e. $\rho_{\max}$ does not exceed some $\rho_{\text{thresh}} < 1$. Wilkinson chose $\rho_{\text{thresh}} = 1/2$, which we use as well for our "cautious" mode.

Even if we start off with geometric convergence, it will cease when the rounding errors $\delta r^{(i)}$ and $\delta x^{(i+1)}$ become large enough. To detect this we have several basic stopping criteria: Thus refinement will

Criterion 1: Stop if $\frac{\|dx^{(i+1)}\|_\infty}{\|x^{(i)}\|_\infty} \le \varepsilon_w$, i.e. if $dx^{(i+1)}$ is too small to change the solution $x^{(i)}$ much, or

Criterion 2: Stop if i.e. if sufficiently fast geometric convergence ceases, or

Criterion 3: Stop if $i > i_{\text{thresh}}$, i.e. if too many iterations have been performed.

Altogether, our normwise error bound is

$$B_{\text{norm}} \overset{\text{def}}{=} \max\left\{\frac{\|dx^{(i+1)}\|_\infty/\|x^{(i)}\|_\infty}{1-\rho_{\max}}, \gamma\varepsilon_w\right\} \approx \frac{\|x^{(i)} - x\|_\infty}{\|x\|_\infty} \overset{\text{def}}{=} E_{\text{norm}} \tag{6}$$

where $\gamma = \max(10, \sqrt{n})$ has been chosen based on our numerical experiments to account for dependence of error on dimension.

Now we discuss diagonal scaling of A. Existing linear equation solvers may already *equilibrate*, i.e. replace the input matrix A by $A_s = R \cdot A \cdot C$ where R and C are diagonal matrices chosen to try to make $\kappa_\infty(A_s)$ much smaller than $\kappa_\infty(A)$. This changes the linear system from $Ax = b$ to $A_s x_s = b_s$ where $x_s = C^{-1}x$ and $b_s = Rb$. If we are careful to choose diagonal entries of R and C to be powers of the floating point radix β, then no roundoff is introduced. The choice of R has no effect on the way we measure error (it effect the pivot choices during factorization), but the choice of C does. We account for this as follows: Iterative refinement on the system $A_s x_s = r_s$ produces a sequence of corrections $dx_s^{(i+1)}$ and approximate solutions $x_s^{(i+1)}$ As long as C's entries are powers of β and we use conventional implementations of LU decomposition (eg not Strassen's method), then $Cdx_s^{(i+1)} = dx^{(i+1)}$ and $Cx_s^{(i+1)} = x^{(i+1)}$ *exactly*, including all rounding errors (but ignoring over/underflow). Therefore we can apply our error bound and stopping criteria above to the quantities $Cdx_s^{(i+1)}$ and $Cx_s^{(i+1)}$. This amounts to $O(n)$ work to diagonally scale before norms are computed, and is very cheap. The corresponding condition number governing convergence is

$$\kappa_{\text{norm}} = \kappa_\infty(A_s \cdot C^{-1}) = \kappa_\infty(R \cdot A). \tag{7}$$

which is easy to estimate using standard condition estimation techniques [9, 10, 11].

Now note that in the last paragraph the matrix C could have contained arbitrary powers of β (modulo over/underflow), not just those from equilibration. Thus if we were able to choose $\hat{C}$ so that each component of $|\hat{C}x_s^{(i+1)}|$ were approximately 1, the *normwise* relative error in $Cx_s^{(i+1)}$ (measure using the infinity norm) is the same as the *componentwise* relative error. Thus all our error bounds, stopping criteria and condition numbers above apply.

Of course we do not how to choose $\hat{C}$ until we know the solution, at least approximately. So in the final algorithm, discussed in the next section, we monitor the convergence of each component of $x_s^{(i+1)}$, and as soon as each component has "settled down" to at least 2 bits (i.e. a relative error of .25), we round it to the nearest power of β in order to determine the corresponding diagonal entry of $\hat{C}$. This leads to the componentwise error bound

$$B_{\text{comp}} \stackrel{\text{def}}{=} \max\left\{\frac{\|\hat{C}dx^{()}(i+1)_s\|_\infty}{1-\rho_{\hat{\max}}}, \gamma\varepsilon_w\right\} \approx \max_k \left|\frac{x_k^{(i)} - x_k}{x_k}\right| \stackrel{\text{def}}{=} E_{\text{comp}} \tag{8}$$

and componentwise condition number

$$\kappa_{\text{comp}} = \kappa_\infty(A_s \cdot \hat{C}^{-1}) \approx \kappa_\infty(A_s \cdot \text{diag}(x_s)) = \kappa_\infty(R \cdot A \cdot \text{diag}(x)). \tag{9}$$

We note that the same technique may be applied to LAPACK's current single precision refinement algorithm in SGERFS. Scaling by $x^{(i)}$ in LAPACK's forward error estimator produces the loose componentwise error estimate $B_{\text{comp}} \approx \|\hat{C}^{-1} \cdot |A^{-1}| \cdot (|r| + (n+1)\varepsilon_w(|A||x_s| + |b|))\,\|_\infty$.

3 Algorithmic Details

Using the error estimates and termination criteria established above, we now describe our ultimate iterative refinement procedure, Algorithm 2. Its cost amounts to a bounded number of triangular solves, on top of Gaussian elimination. In other words, for a dense matrix, the cost is $O(n^2)$ on top of the $O(n^3)$ required for the solution without refinement.

The algorithm refers to auxiliary vectors $y \equiv x_s$, $dy \equiv dx_s$, $z \equiv \hat{C}x_s$ and $dz \equiv \hat{C}dx_s$. but the implementation stores y in place of x and dy in places of dx to minimize storage; neither z nor dz are stored explicitly. All norms in this section are the infinity norm.

Algorithm 2 tracks the convergence state of x and z in the variables x-state $\in$ {working, no-progress, converged} and z-state $\in$ {unstable, working, no-progress, converged}, respectively. State variable y-scheme $\in$ {single, double} denotes the precision used for storing and computing with y. The refinement loop exits either from a large iteration count or when both x and z are no longer in a working state (see line 18).

The logic is somewhat complicated by three facts. First, we are simultaneously applying two sets of stopping criteria, one for the normwise error and one for componentwise error, so the code must decide what to do when one set of criteria decide to stop (either because of convergence or failure to converge) before the other set of criteria; basically we continue if are continuing to make progress by either set of criteria.

Second, the code must decide when store the solution y in doubled working precision $\varepsilon_x \le \varepsilon_w^2$; this costs more than keeping y in working precision, and so is only used when either progress toward a solution stops (Criterion 2) before convergence (Criterion 1) (in either the normwise or componentwise sense), or when a cheap upper bound on κ_{comp} is very large (line 14). To store y in doubled working precision a second vector y_t is used to store the "tails" of each floating point component. In otherwords, the solution is represented as $y + y_t$ where $|y_t| \ll |y|$.

Third, we must decide when each component of the solution has "settled down" enough to test for componentwise relative convergence: Our criterion is that no component's relative change exceeds a threshold dz_{thresh}, currently set to .25.

There are a number of slightly different ways to implement all the above tests. As described in [7], we tried a variety of formulations, because in our 6.2 million tests there were noticeable statistical differences in the number of "outliers", where the error was larger than desired. We chose the formulation that minimized these outliers.

We must also take precautions to return reasonable error bounds when some solution components are exactly zero because of the sparsity structure of A and b. For example, if $b = 0$ then $x = 0$ should be returned with a zero error bound. Similarly, if A is block diagonal and b has zero components then some components of x will be exactly zero. See [7, Sec. 7.3] for further discussion.

```
Input: Current z-state, $\|dz^{(i)}\|$, $\|dz^{(i-1)}\|$, y-scheme
Output: New z-state and $\rho_{\text{max,z}}$, possibly signaling incr-prec or updating final-relnorm$_z$
if z-state = unstable and $\|dz^{(i+1)}\| \le dz_{\text{thresh}}$ then z-state = working
if z-state = no-progress and $\|dz^{(i+1)}\|/\|dz^{(i)}\| \le \rho_{\text{thresh}}$ then z-state = working
if z-state = working then
    if $\|dz^{(i+1)}\| \le \varepsilon_w$ then z-state = converged // Criterion 1, tiny $dz$
5   else if $\|dz^{(i+1)}\| > dz_{\text{thresh}}$ then
        z-state = unstable, final-relnorm$_z = \infty$, $\rho_{\text{max,z}} = 0.0$
    else if $\|dz^{(i+1)}\|/\|dz^{(i)}\| > \rho_{\text{thresh}}$ then
        if y-scheme = single then incr-prec = true
        else z-state = no-progress // Criterion 2, lack of progress
    else $\rho_{\text{max,z}} = \max\{\rho_{\text{max,z}}, \|dz^{(i+1)}\|/\|dz^{(i)}\|\}$
    if z-state $\ne$ working then final-relnorm$_z = \|dz^{(i+1)}\|$
```

Procedure `new-z-state`

4 Related Work

Extra precise iterative refinement was proposed in the 1950s [22] and analyzed in the 1960s. In [6], Wilkinson *et al.* presents the Algol programs that perform the LU factorization, the triangular solutions, and the iterative refinement using $\varepsilon_r = \varepsilon_w^2$. Wilkinson used the Crout algorithm for LU factorization, in which the inner products are performed in extra precision, but in our numerical experiment we use the same implementation of Gaussian elimination from LAPACKas Algorithm 2.

Wilkinson's algorithm differs from ours in several ways: (1) There is no initial equilibration in Wilkinson's algorithm. (2) ρ_{thresh} is fixed to 0.5 in Wilkinson's algorithm. (3) Wilkinson's algorithm does not store x to additional precision. (4) Wilkinson's algorithm does not attempt to achieve componentwise accuracy. (5) The original paper's algorithm [6] does not return an error bound,

```
   Input: An $n \times n$ matrix $A$, an $n \times 1$ vector $b$
   Output: A solution vector $x^{(i)}$ approximating $x$ in $Ax = b$,
           a normwise error bound $\approx \|x^{(i)} - x\| / \|x\|$, and
           a componentwise error bound $\approx \max_k |x_k^{(i)} - x_k| / |x_k|$
   Equilibrate the system: $A_s = R \cdot A \cdot C$, $b_s = R \cdot b$
   Estimate $\kappa_{\text{s}} = \kappa_\infty(A_s)$
   Solve $A_s y^{(1)} = b_s$ using the basic solution method
 4 $\|dx^{(1)}\| = \|dz^{(1)}\| = \textsf{final-relnorm}_x = \textsf{final-relnorm}_z = \infty$
   $\rho_{\max,\text{x}} = \rho_{\max,\text{z}} = 0.0$,   x-state = working,   z-state = unstable,   y-scheme = single
 6 for $i = 1$ to $i_{\text{thresh}}$ do
       // Compute residual in precision $\varepsilon_r$
       if y-scheme = single then $r^{(i)} = A_s y^{(i)} - b_s$
       else $r^{(i)} = A_s(y^{(i)} + y_t^{(i)}) - b_s$, using doubled arithmetic
       // Compute correction to $y^{(i)}$
       Solve $A_s\, dy^{(i+1)} = r^{(i)}$ using the basic solution method
       // Check error-related stopping criteria
       Compute $\|x^{(i)}\| = \|Cy^{(i)}\|$, $\|dx^{(i+1)}\| = \|Cdy^{(i+1)}\|$ and $\|dz^{(i+1)}\| = \max_j \left| dy_j^{(i+1)} / y_j^{(i)} \right|$
14     if y-scheme = single and $\kappa_{\text{s}} \cdot \max_j |y_j| / \min_j |y_j| \geq 1/\gamma\varepsilon_w$ then incr-prec = true
15     Update x-state, $\rho_{\max,\text{x}}$ with Procedure new-x-state below
16     Update z-state, $\rho_{\max,\text{z}}$ with Procedure new-z-state below
       // Either update may signal incr-prec or may set its final-relnorm .
18     if x-state $\neq$ working and z-state $\neq$ working then BREAK
19     if incr-prec then y-scheme = double, incr-prec = false, and $y_t^{(i)} = 0$
       // Update solution
       if y-scheme = single then $y^{(i+1)} = y^{(i)} - dy^{(i+1)}$
       else $(y^{(i+1)} + y_t^{(i+1)}) = (y^{(i)} + y_t^{(i)}) - dy^{(i+1)}$ in doubled arithmetic
23 if x-state = working then $\textsf{final-relnorm}_x = \|dx^{(i+1)}\| / \|x^{(i)}\|$
24 if z-state = working then $\textsf{final-relnorm}_z = \|dz^{(i+1)}\|$
   return $x^{(i)} = Cy^{(i)}$,
       normwise error bound $\max\left\{ \frac{1}{1-\rho_{\max,\text{x}}} \cdot \textsf{final-relnorm}_x, \max\{10, \sqrt{n}\} \cdot \varepsilon_w \right\}$, and
       componentwise error bound $\max\left\{ \frac{1}{1-\rho_{\max,\text{z}}} \cdot \textsf{final-relnorm}_z, \max\{10, \sqrt{n}\} \cdot \varepsilon_w \right\}$
```

Algorithm 2: New iterative refinement

```
Input: Current x-state, $\|x^{(i)}\|$, $\|dx^{(i)}\|$, $\|dx^{(i-1)}\|$, y-scheme
Output: New x-state and $\rho_{\max,\text{x}}$, possibly signaling incr-prec or updating $\textsf{final-relnorm}_x$
if x-state = no-progress and $\|dx^{(i+1)}\| / \|dx^{(i)}\| \leq \rho_{\text{thresh}}$ then x-state = working
if x-state = working then
    if $\|dx^{(i+1)}\| / \|x^{(i)}\| \leq \varepsilon_w$ then x-state = converged // Criterion 1, tiny $dx$
    else if $\|dx^{(i+1)}\| / \|dx^{(i)}\| > \rho_{\text{thresh}}$ then
        if y-scheme = single then incr-prec = true
        else x-state = no-progress // Criterion 2, lack of progress
    else $\rho_{\max,\text{x}} = \max\{\rho_{\max,\text{x}}, \|dx^{(i+1)}\| / \|dx^{(i)}\|\}$
    if x-state $\neq$ working then $\textsf{final-relnorm}_x = \|dx^{(i+1)}\| / \|x^{(i)}\|$
```

Procedure `new-x-state`

though an error analysis appears in [26] and [18]. In particular, [18, Eq. (11)] and [12, Thm. 12.1] provide error bounds including several functions of problem dimension, condition number, and other quantities that are hard to estimate tightl and efficiently. For the sake of later numerical comparisons, we add normwise error bounds and stopping criteria to Wilkinson's algorithm based on our error analysis in Section 2, and call the resulting algorithm Algorithm W.

The use of higher precision in computing x was first presented as an exercise in Stewart's book [23, p. 206-207]. Stewart suggests that if x is accumulated in higher and higher precision, then the residual will get progressively smaller and the iteration will eventually converge to any desired accuracy. Kiełbasiński proposes an algorithm called binary cascade iterative refinement [14]. In this algorithm, GEPP and the first triangular solve for $x^{(0)}$ are performed in a base precision. Then during iterative refinement, $r^{(i)}$, $x^{(i+1)}$ and $dx^{(i)}$ are computed in successively higher precision. Kiełbasiński's shows that with a prescribed accuracy for x, you can choose a maximum precision required to stop the iteration. This requires arbitrary precision arithmetic. We are not aware of any implementation of this algorithm.

A different approach to guaranteeing accuracy of a solution is to use interval arithmetic [20, 21]. We will not consider interval algorithms further, since the available algorithms do not meet our criterion of costing a small amount extra on top of the fastest available solution method.

Björck [3] nicely surveys the iterative refinement for linear systems and least-squares problems, including error estimates using working precision or extra precision in residual computation. Higham's book [12] gives a detailed summary of various iterative refinement schemes which have appeared through history. Higham also provides estimates of the limiting normwise and componentwise error. The estimates are not intended for computation but rather to provide intuition on iterative refinement's behavior. The estimates involve quantities like $\| |A^{-1}| \cdot |A| \cdot |x| \|_\infty$ and require estimating the norm of A^{-1}. We experimented with approximating these error estimates without using refinement within the norm estimator (which requires the solution of linear systems with A and A^T), but they provided very inaccurate normwise and componentwise bounds.

Until now, extra precise iterative refinement was not adopted in standard libraries, such as LINPACK [8] and later LAPACK [1], mainly because there was no portable way to implement extra precision when the working precision was already the highest precision supported by the compiler. Therefore, the current LAPACK expert driver routines xGESVX only provide the working precision iterative refinement routines ($\varepsilon_r = \varepsilon_w$). Since iterative refinement can always ensure backward stability, even in working precision [12, Theorem 12.3], the LAPACK refinement routines use the componentwise backward error in the stopping criteria. For the sake of later numerical comparisons, we augment this algorithm as described in Section 2 to compute a componentwise error bound, and call the resulting algorithm Algorithm L.

5 Testing Configuration

5.1 Hardware and Software Platforms.

The XBLAS library [15] is a set of routines for dense and banded BLAS routines, along with their extended and mixed precision versions; see Chapters 2 and 4 of the BLAS Technical Forum Standard [5]. Many routines in the XBLAS library allow higher internal precision to be used, enabling us to compute more accurate residuals. For example, general matrix-vector multiply `BLAS_sgemv_x` performs $y \leftarrow \alpha A x + \beta y$ in single, double, indigenous, or extra precision.

In addition to the extra precision routines provided by the XBLAS, the doubled-x scheme in Algorithm 2 requires a new routine which we call `gemv2`. This routine takes a matrix A, three vectors x, y, and z, and three scalars α, β, and γ to compute $z \leftarrow \alpha Ax + \beta Ay + \gamma z$ where the right hand side is evaluated with precision ε_r. This routine enables us to compute an accurate residual when x is kept in two words, x_{leading} and x_{trailing}: $r = b - A(x_{\text{leading}} + x_{\text{trailing}})$.

We tested the code on two platforms: Sun UltraSPARCs running Solaris and Itanium 2 running Linux. The results were largely similar, so we present results only from Itanium. For more details regarding compilers and BLAS libraries used, see [7, Sec. 5.5].

5.2 Test Matrix Generation

To thoroughly test our algorithms, we need many test cases with a wide range of condition numbers, various distribution of singular values, well-scaled and ill-scaled matrices, matrices with first k columns nearly linearly dependent (so that ill-conditioning causes the k-th pivot to be tiny), and a wide range of solution component sizes. We generate test cases as follows:

1. Randomly pick a condition number κ with $\log_2 \kappa$ distributed uniformly in $[0, 26]$. This will be the (2-norm) condition number of the matrix before any scaling is applied.

2. We pick a set of singular values σ_i's from one of the following choices:

   (a) One large singular value: $\sigma_1 = 1$, $\sigma_2 = \cdots = \sigma_n = \kappa^{-1}$.

   (b) One small singular value: $\sigma_1 = \sigma_2 = \cdots = \sigma_{n-1} = 1$, $\sigma_n = \kappa^{-1}$.

   (c) Geometrical distribution: $\sigma_i = \kappa^{-\frac{i-1}{n-1}}$ for $i = 1, 2, \ldots, n$.

   (d) Arithmetic distribution: $\sigma_i = 1 - \frac{i-1}{n-1}(1 - \kappa^{-1})$ for $i = 1, 2, \ldots, n$.

3. Pick k randomly from $\{3, n/2, n\}$. Move the largest and the smallest singular values (picked in step 1) into the first k values, and let Σ be the resulting diagonal matrix. let

$$\tilde{A} = U\Sigma \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \tag{10}$$

   where U, V_1, and V_2 are random orthogonal matrix with dimensions n, k, and $n-k$, respectively. If κ is large, this makes the leading k columns of $\tilde{A}$ nearly dependent, so that LU factorization will encounter a small pivot at the k-th step. U, V_1 and V_2 are applied via sequences of random reflections of dimensions 2 through n, k or $n-k$, resp.

4. Pick τ with $(\log_2 \tau)^{1/2}$ uniformly distributed in $[0, \sqrt{24}]$. We generate $\tilde{x}$ so that $\tau = \frac{\max_i |\tilde{x}_i|}{\min_i |\tilde{x}_i|}$. by randomly choosing one of the following distributions:

   (a) One large component: $\tilde{x}_1 = 1$, $\tilde{x}_2 = \cdots = \tilde{x}_n = \tau^{-1}$.

   (b) One small component: $\tilde{x}_1 = \tilde{x}_2 = \cdots = \tilde{x}_{n-1} = 1$, $\tilde{x}_n = \tau^{-1}$.

   (c) Geometrical distribution: $\tilde{x}_i = \tau^{-\frac{i-1}{n-1}}$ for $i = 1, 2, \ldots, n$.

   (d) Arithmetic distribution: $\tilde{x}_i = 1 - \frac{i-1}{n-1}(1 - \tau^{-1})$ for $i = 1, 2, \ldots, n$.

   (e) $\tilde{x}_i$'s are randomly chosen within $[\tau^{-1}, 1]$ so that $\log \tilde{x}_i$ is uniformly distributed.

If one of the first four distributions is chosen, we multiply $\tilde{x}$ by a uniform random number in $[0.5, 1.5]$ to make the largest element unequal to 1 (so that all components of $\tilde{x}$ have full significand).

5. We then randomly column scale the matrix $\tilde{A}$ generated in step 3. We pick a scaling factor δ such that $(-\log_2 \delta)^{1/2}$ is uniformly distributed in $[0, \sqrt{24}]$.* Select two columns of $\tilde{A}$ at random and multiply by δ to produce the final input matrix A (rounded to single).

6. We compute $b = A\tilde{x}$ using double-double precision (using the XBLAS routine `BLAS_sgemm_x`), but rounded to single at the end.

7. Compute $x = A^{-1}b$ by using double precision GEPP with double-double precision iterative refinement. This corresponds to Algorithm 2 with IEEE754 double precision as working precision ($\varepsilon_w = 2^{-53}$) and double-double as residual precision ($\varepsilon_r \approx 2^{-105}$). Note that the "true" solution x thus obtained is usually not the same as $\tilde{x}$ that we started, due to rounding errors committed in step 6. This difference can be quite large if A is ill-conditioned.

Using the above procedure, 2×10^6 5×5, 2×10^6 10×10, 2×10^6 100×100, 2×10^5 1000×1000 test matrices were generated. Statistics for the 100×100 matrices are presented next. Similar results were obtained for the other dimensions.

5.3 Test Matrix Statistics

We histogrammed 6 different condition numbers of the test matrices to make sure that we had a good (if not entirely uniform) sampling of hard through easy problems. The histograms may be seen in [7, Fig. 1]. These condition numbers are as follows.

1. $\kappa_{\mathrm{s}} = \kappa_\infty(RAC) = \kappa_\infty(A_s)$ is the condition number of the equilibrated system. This is a measure of the inherent difficulty of the system (measured in an appropriate norm). This condition number varies from about from about 57 to 1.6×10^{13}, with all but 2169 (0.11%) smaller than 10^{10}.

2. $\kappa_{\mathrm{c}} = \kappa_\infty(C) = \|C\|_\infty$ is the maximum column scaling factor computed during equilibration. This varies from 1 to $2^{35} \approx 3.4 \times 10^{10}$. Since the equilibration is only done if $\|C\|$ would be greater than 10, $C = I$ happens relatively often.

3. $\kappa_{\mathrm{x}} = \kappa_\infty(\mathrm{diag}(x)) = {}^{\max_i |x_i|}/_{\min_i |x_i|}$ is the ratio between the largest and smallest element (in magnitude) of x. This is a measure of how wildly the components of x varies. κ_{x} varies from 1 to about 6.8×10^{13}, with all but 750 (0.04%) of them less than 10^{10}.

4. $\kappa_{\mathrm{y}} = \kappa_\infty(\mathrm{diag}(y)) = {}^{\max_i |y_i|}/_{\min_i |y_i|}$ is the ratio between the largest and smallest element (in magnitude) of y. This varies from 1 to approximately 8.5×10^{12}, with all but 1266 (0.06%) less than 10^{10}. This is a measure of how wildly the components of y varies, which gives some idea of the difficulty of getting componentwise accurate solution. The term κ_{y} appears naturally in the condition number for componentwise problem, since $\kappa_{\mathrm{comp}} = \kappa_\infty(A_s\,\mathrm{diag}(y)) \le \kappa_{\mathrm{s}}\kappa_{\mathrm{y}}$.

*As in step 4, we chose this distribution as to not overload our test samples with "hard" problems (in normwise sense) with large $\kappa(RA) = \kappa(A_s C^{-1})$.

5. $\kappa_{\text{norm}} = \kappa_\infty(A_s \cdot C^{-1}) = \kappa_\infty(R \cdot A)$ is the normwise condition number. It varies from about 57 to 7.6×10^{17}. Using κ_{norm} we divide the test matrices into two categories:

   - **Normwise "well"-conditioned problems**. These are matrices with $\kappa_{\text{norm}} \leq {}^{1}\!/\!{}_{\gamma\varepsilon_w}$, and perhaps more accurately described as "not too ill-conditioned" matrices. These are matrices where we hope to have an accurate solution after refinement in the normwise sense. Most problems can be expected to fall in this category in practice. Of the two million test matrices, 821097 cases fall into this category.
   - **Normwise ill-conditioned problems**. These are the matrices with $\kappa_{\text{norm}} > {}^{1}\!/\!{}_{\gamma\varepsilon_w}$. These are so ill-conditioned that we cannot guarantee accurate solutions. Of the two million test matrices, 1178903 cases are in this category.

   Note that the choice of ${}^{1}\!/\!{}_{\gamma\varepsilon_w}$ (which is approximately 1.67×10^6 for 100×100 matrices) as the separation between well and ill-conditioned matrices is somewhat arbitrary; we could have chosen a more conservative criteria such as ${}^{1}\!/\!{}_{n\varepsilon_w}$ or more aggressive criteria such as ${}^{1}\!/\!{}_{\varepsilon_w}$. Our data in Section 6 indicate that the choice ${}^{1}\!/\!{}_{\gamma\varepsilon_w}$ seems to give reliable solutions without throwing away too many convergent cases. $\gamma = \max\{10, \sqrt{n}\}$ both protects against condition number underestimates and keeps the bounds attainable.

6. $\kappa_{\text{comp}} = \kappa(RA \operatorname{diag}(x)) = \kappa(A_s \operatorname{diag}(y))$ is the condition number for componentwise problem. It varies from about 57 to 4.5×10^{15}. Note that κ_{comp} depends not only on the matrix A, but also on the right hand side vector b (since it depends on the solution x). As before, we use κ_{comp} to divide the problems into two categories:

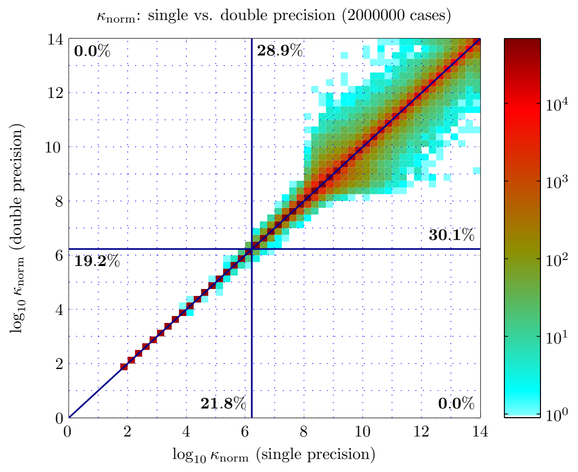
   - **Componentwise "well"-conditioned problems**. These are problems with $\kappa_{\text{comp}} \leq {}^{1}\!/\!{}_{\gamma\varepsilon_w}$, and perhaps more accurately described as "not too ill-conditioned" problems. These are problems where we hope to have an accurate solution in componentwise sense. Of the two million test problems, 545427 cases fall into this category.
   - **Componentwise ill-conditioned problems**. These are the matrices with $\kappa_{\text{comp}} > {}^{1}\!/\!{}_{\gamma\varepsilon_w}$. These are so ill-conditioned that we cannot guarantee accurate solutions. Of the two million test problems, 1454573 cases fall into this category. Note that if any component of the solution is zero, then κ_{comp} becomes infinite.

We also histogrammed κ_s versus κ_c and κ_s versus κ_y (see [7, Fig. 2]) in order to confirm that we sampled the test matrix space thoroughly.

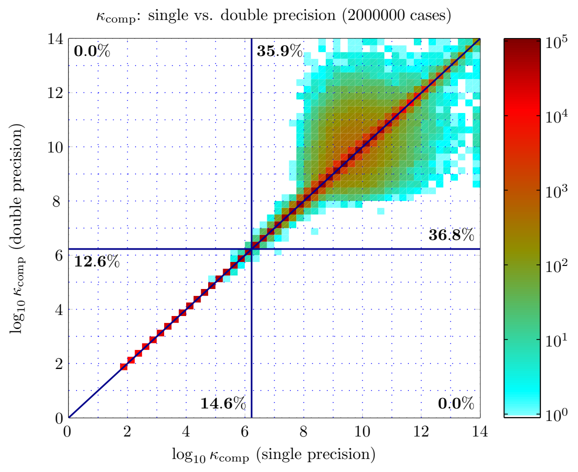
For the basic solution method, we used LAPACK's `sgetrf` (GEPP, $PA_s = LU$) and `sgetrs` (triangular solve) routines. We observed that the pivot growth factors $\max_{i,j} |U(i,j)| / \max_{i,j} |A_s(i,j)|$ are no more than 72. This implies that we have obtained LU factors with small normwise backward error. We histogrammed the pivot growth factors (see [7, Fig. 3]), which showed a smooth decrease in likelihood with increasing pivot growth. This is consistent with [25], which suggests that for various random matrix distributions, the average pivot growth factor should be about $n^{2/3} \approx 22$.

5.4 Accuracy of Single Precision Condition Numbers

All the condition numbers in section 5.3 were computed in double precision, which can be regarded as truth. Since the Algorithm 2 will only estimate the condition number in working (single) precision, we must confirm that single precision result is accurate for condition numbers up to ${}^{1}\!/\!{}_{\gamma\varepsilon_w}$.



(a) κ_{norm}



(b) κ_{comp}

Figure 1: Accuracy of computed condition numbers: single precision vs. double precision. The single precision κ_{comp} are those computed by Algorithm 2 with $\rho_{\text{thresh}} = 0.5$.

The 2D histograms of the normwise condition number κ_{norm} computed in single precision and double precision, is shown in figure 1(a). Since 2D histograms appear throughout this paper, we explain this plot in detail. In this 2D histogram, each colored square at coordinate (x, y) indicates the existence of matrices that have single precision κ_{norm} in the range $[10^x, 10^{x+1/4})$ and double precision κ_{norm} in the range $[10^y, 10^{y+1/4})$. The color of each square indicates the number of matrices that fall in that square, indicated by the color bar to the right of the plot. Red (dark) colored squares indicate large population while cyan (light) colored squares indicate very small population. A logarithmic scale is used in the color bar, so a lighter color indicates a much smaller population than a darker color. For example, the light cyan colored squares near the top and right edges contains only a handful of matrices (less than 5, usually just 1), while the darker red colored squares contains approximately 10^3 to 10^4 samples.

The blue horizontal and vertical lines are both located at $1/\gamma\varepsilon_w$, separating "well"-conditioned" and ill-conditioned matrices. Along with the diagonal line where both single and double precision values of κ_{norm} are equal, these lines divide the plot into six regions, and the percentage of samples in each region is displayed in bold numbers in the plot.

If the single and double precision values of κ_{norm} were identical, the only colored squares would be along the diagonal. When $\kappa_{\text{norm}} \leq 1/\gamma\varepsilon_w$, the squares are all close to the diagonal, meaning that the single precision κ_{norm} is close to the double precision κ_{norm} (which we assume is the true value). Only when either condition number exceeds $1/\gamma\varepsilon_w$ can they differ significantly, as indicated by the spread of colored squares in the upper right. This tells us that we can trust the single precision κ_{norm} to separate the not-too-ill-conditioned matrices from the very ill-conditioned matrices.

Figure 1(b) tells the same story for the componentwise condition number κ_{comp}.

6 Numerical Results

We present the numerical results for our new algorithm, Algorithm 2, and compare it to Algorithms W (Wilkinson) and L (the current LAPACKalgorith, with our small modifications). The data is for two million 100×100 test matrices described in Section 5. Similar results were obtained on two million each of 5×5 and 10×10 matrices and also two hundred thousand 1000×1000 matrices[*].

The normwise and componentwise true errors are denoted $E_{\text{norm}} = {\|x-\hat{x}\|_\infty}/{\|x\|_\infty}$ and $E_{\text{comp}} = \max_i {|x_i - \hat{x}_i|}/{|x_i|}$, respectively. where x is the true solution and $\hat{x}$ is the solution computed by the algorithm. Similarly, the normwise and componentwise error bounds (computed by the algorithms) are denoted B_{norm} and B_{comp}, respectively.

Sections 6.1 and 6.2 present normwise and componentwise results, resp. We set $\rho_{\text{thresh}} = 0.5$ in Algorithms 2 and W. Parameter i_{thresh} (maximum number of iterations allowed) is set very large so that we can evaluate the number of steps required to converge.

We define notation common to Sections 6.1 and 6.2. For example, for E_{norm} we distinguish three cases:

1. **Strong Convergence**: $E_{\text{norm}} \le 2\gamma\varepsilon_w = 2\max\{10, \sqrt{n}\} \cdot \varepsilon_w$. This is the most desirable case, where the true error is of order ε_w. The lower solid horizontal blue line in Figure 2 (and in other analogous figures) is at $E_{\text{norm}} = 2\gamma\varepsilon_w$.

2. **Weak Convergence**: $2\gamma\varepsilon_w < E_{\text{norm}} \le \sqrt{\varepsilon_w}$. We could not get strong convergence, but we did get at least half of the digits correctly. The upper solid horizontal blue line in Figure 2 (and in other analogous figures) is at $E_{\text{norm}} = \sqrt{\varepsilon_w}$.

3. **No Convergence**: $E_{\text{norm}} > \sqrt{\varepsilon_w}$. We could not get a meaningful result.

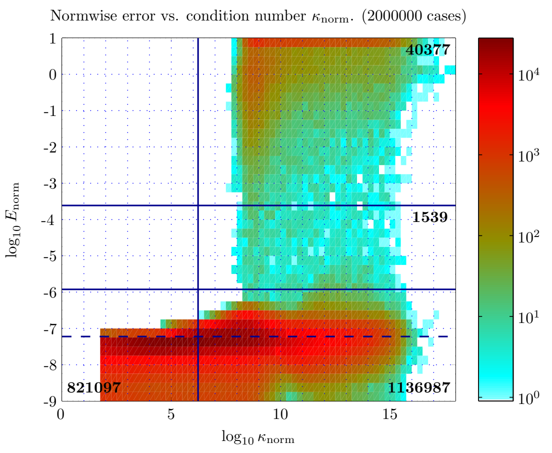
In addition, we often indicate the value of ε_w in figures as well. For example the dashed horizontal blue line in Figure 2 (and in other analogous figures) is at $E_{\text{norm}} = \varepsilon_w$.

Many other analogous figures have two solid horizontal and one dashed horizontal line to indicate similar thresholds betreen these cases for B_{norm}, E_{comp}, and B_{comp}.

In the final version of the code we set the error bound to 1 whenever its computed value exceeds $\sqrt{\varepsilon_w}$, in order to indicate that it did not converge according to our criterion above. But in this section we report the computed error bounds without setting them to 1, in order to better understand their behavior.

Later in Section 6.4 we will vary the parameters ρ_{thresh} and i_{thresh} to study how the behavior of Algorithm 2 changes. In particular, we will recommend "cautious" and "aggressive" values of i_{thresh} and ρ_{thresh}. The cautious settings, which we recommend as the default, yield maximally reliable error bounds for "well"-conditioned problems, and cause the code to report convergence failure on the hardest problems. The aggressive settings will lead to more iterations on the hardest problems and usually, but not always, give error bounds within a factor of 100 of the true error.

[*]A full set of plots for all algorithms and all 6.2M test matrices is at `http://www.cs.berkeley.edu/~demmel/itref-data/`.



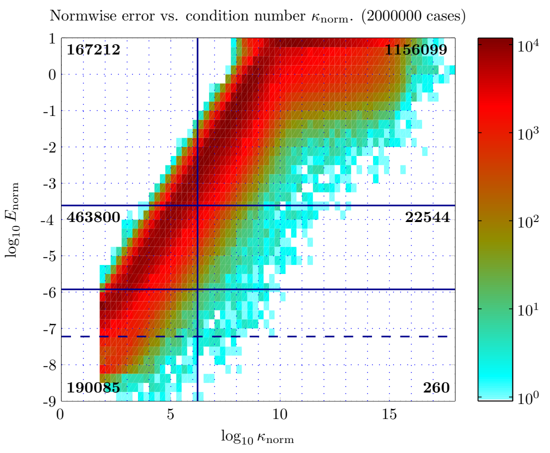


Figure 2: Normwise error vs. κ_{norm}. Left: Algorithm 2 with $\rho_{\text{thresh}} = 0.5$. Right: Algorithm L.

6.1 Normwise Error Estimate

The most important observation is that both Algorithm 2 and Algorithm W deliver a tiny error (E_{norm} strongly converged) and a slightly larger error bound (B_{norm} also strongly converged) as long as $\kappa_{\text{norm}} \leq {}^{1}\!/\gamma\varepsilon_w$, i.e. for all "well"-conditioned matrices in our test set (821097 out of 2 million). This is the best possible behaviour we could expect.

The second important observation is that for ill-conditioned problems ($\kappa_{\text{norm}} > {}^{1}\!/\gamma\varepsilon_w$), both algorithms still do very well, with Algorithm 2 getting strong convergence in both E_{norm} and B_{norm} in 96.4% of the cases, and Algorithm W getting strong convergence in both E_{norm} and B_{norm} in 92.3% of the cases.

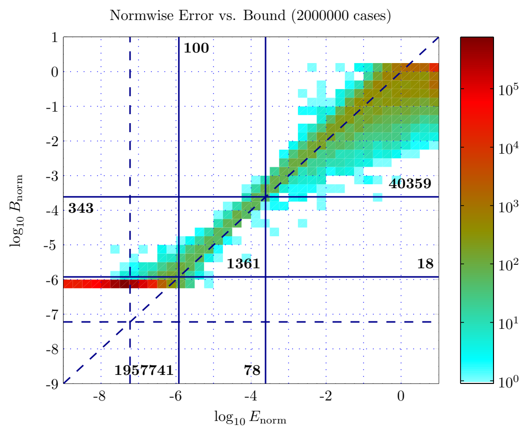
In the rest of this section, we explore the experimental data in more detail, describing what goes wrong when we fail to get strong convergence.

The two plots in Figure 2 show the 2D histograms of the test problems plotted according to their true normwise error E_{norm} and condition number κ_{norm} for Algorithms 2 and L. The plot for Algorithm W is nearly identical to the plot for Algorithm 2 and so is omitted (see [7, Fig. 6b]). The vertical solid line is at $\kappa_{\text{norm}} = {}^{1}\!/\gamma\varepsilon_w$, and separates the "well"-conditioned problems from the ill-conditioned problems. The solid horizontal lines are at $\sqrt{\varepsilon_w}$ and $2\gamma\varepsilon_w$, separating the regions of strong and weak convergence of E_{norm}. The dotted horizontal line is at ε_w, far below which errors are unlikely to fall. If any problem falls outside the coordinate range, then it is placed at the edge; for example the band at the very top of Figure 2 includes all the cases where $E_{\text{norm}} \geq 10$.

The first important conclusion to draw from the left figure in Figure 2 is that for "well"-conditioned problems ($\kappa_{\text{norm}} < {}^{1}\!/\gamma\varepsilon_w$), Algorithm 2 (and Algorithm W) attain the best possible result: strong convergence of E_{norm} in all cases (821097 out of 2 million).

The second important conclusion is that for harder problems, with $\kappa_{\text{norm}} \geq {}^{1}\!/\gamma\varepsilon_w$ (1178903 cases) Algorithm 2 (and Algorithm W) still do very well, with Algorithm 2 exhibiting strong convergence of E_{norm} in 96.4% of cases (1136987 out of 1178903), and Algorithm W exhibiting strong convergence of E_{norm} in 93.3% of cases (1100328 out of 1178903).

In contrast, with Algorithm L (the right figure in Figure 2), the error grows roughly proportional to the condition number, as shown by the dark diagonal squares in the figure. This is consistent



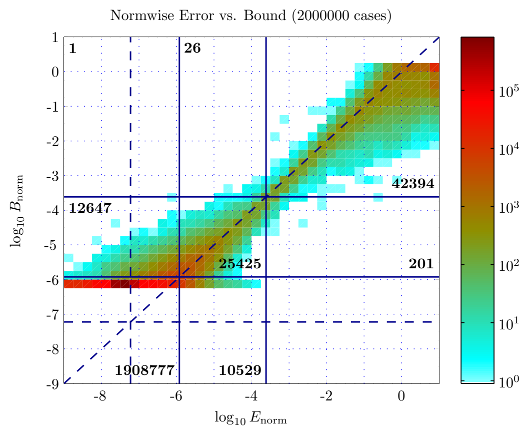


Figure 3: Normwise error vs. bound (all two million cases) Left: Algorithm 2. Right: Algorithm W.

with the early error analysis on the working precision iterative refinement [12, Theorem 12.2]. Algorithm L consistently gives much larger true errors and error bounds than either of the other two algorithms, when it converges.

But of course a small error E_{norm} is not helpful if the algorithm cannot recognize it by computing a small error bound B_{norm}. We now compare B_{norm} to E_{norm} to see how well our error estimate approximates the true error. Figure 3 shows the value of $(E_{\text{norm}}, B_{\text{norm}})$ as a 2D histogram, for Algorithms 2 and W. The leftmost vertical solid blue lines are at $E_{\text{norm}} = 2\gamma\varepsilon_w$ (corresponding to the threshold for strong convergence) and the rightmost vertical solid blue lines are at $E_{\text{norm}} = \sqrt{\varepsilon_w}$ (corresponding to the threshold for weak convergence). Horizontal blue lines are at B_{norm} equal to the same values. The diagonal line marks where B_{norm} is equal to E_{norm}; matrices below the diagonal correspond to *underestimates* ($B_{\text{norm}} < E_{\text{norm}}$), and matrices above the diagonal correspond to *overestimates* ($B_{\text{norm}} > E_{\text{norm}}$). The vertical and horizontal dotted lines correspond to $E_{\text{norm}} = \varepsilon_w$ and $B_{\text{norm}} = \varepsilon_w$, respectively.

For the 821097 "well"-conditioned problems, both Algorithms 2 and W converged to a true error of at most about $3 \cdot 10^{-7}$, and returned an error bound of $\gamma\varepsilon_w \approx 6 \cdot 10^{-7}$, just slightly larger. All these cases appear at the left of the dark band at the bottom left of the histogram, within the region of strong convergence for both error bound and true error. The dark red squares at the bottom of the histograms indicate that the vast majority of these cases returned an true error around ε_w. Thus we can trust either algorithm to deliver a tiny error and a slightly larger error bound as long as $\kappa_{\text{norm}} < 1/\gamma\varepsilon_w$.

By subtracting out the 821097 "well"-conditioned cases from the over 1.9M in the lower left corner of the plots in Figure 3, we get the distribution of results $(E_{\text{norm}}, B_{\text{norm}})$ for all ill-conditioned cases. Most still yield strong convergence in both quantities, but it is interesting to contrast the behavior of Algorithms 2 and W.

The first impression from these plots is that among these ill-conditioned cases, both Algorithms 2 and W fail to have B_{norm} converge in about same number of cases: 3.4% and 3.6% repectively. But of the remainder, those where both algorithms report an error bound to the user, Algorithm 2 fails to get strong convergence in both E_{norm} and B_{norm} in only 0.16% of the cases (1800 out of 1138444 cases), whereas Algorithm W fails to get strong convergence in both E_{norm} and B_{norm} over 27 times

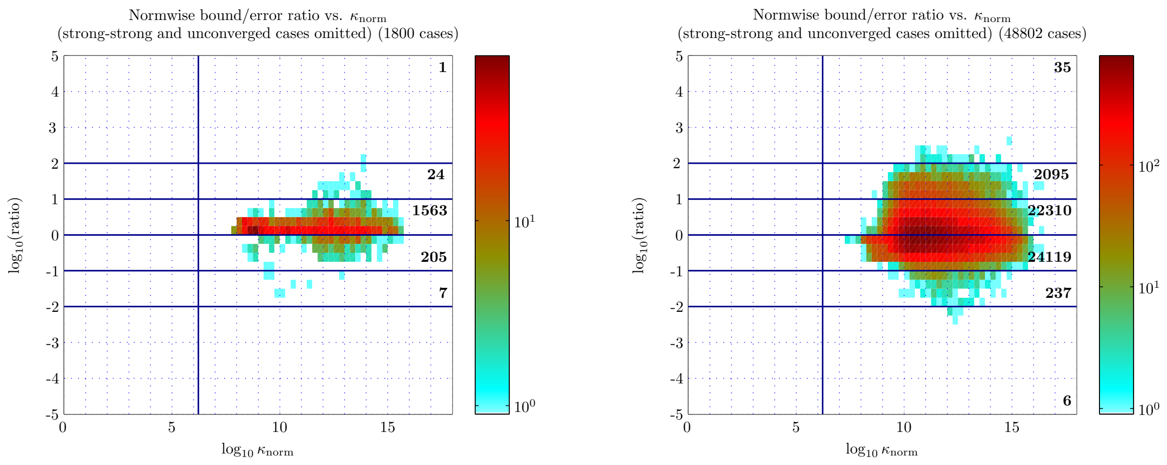


Figure 4: Overestimation and underestimation ratio ($^{B_{\text{norm}}}/_{E_{\text{norm}}}$) vs. κ_{norm}. Cases with strong convergence (in both E_{norm} and B_{norm}) and cases with no convergence ($B_{\text{norm}} > \sqrt{\varepsilon_w}$) are omitted for clarity.
Left: Algorithm 2 with $\rho_{\text{thresh}} = 0.5$. Right: Algorithm W.

as often, in 4.3% of the cases (48802 out of 1136482). In other words, there are over 27 times more cases where Algorithm W is "confused" than Algorithm 2 (48802 versus 1800), and may return significantly disagreeing E_{norm} and B_{norm}.

Finally, Figure 4 shows the 2D histogram of the ratio $^{B_{\text{norm}}}/_{E_{\text{norm}}}$ plotted against κ_{norm}. These plots show how much B_{norm} overestimates E_{norm} (ratio > 1) or underestimates E_{norm} (ratio < 1). We omit cases where B_{norm} does not converge, and also cases where both E_{norm} and B_{norm} converged strongly (the ideal case), since we are only interested in analyzing in cases where the algorithm claims to converge to a solution but either E_{norm} or B_{norm} or both are much larger than ε_w. Since Algorithms 2 and W converge strongly for both E_{norm} and B_{norm} for all "well"-conditioned cases, no data points appear left of the vertical line in the left or right of Figure 4. We are most concerned about underestimates, where the error bound is substantially smaller than the true error. We see that Algorithm 2 has somewhat fewer underestimates than Algorithm W (defined as $E_{\text{norm}} > 10B_{\text{norm}}$), 7 vs. 243, and rather fewer overestimates ($10E_{\text{norm}} < B_{\text{norm}}$), 25 vs. 2130.

Analogous figures for Algorithm L (see [7, Figs. 6c and 8c]) indicate that while Algorithm L never underestimates the error, it almost always overestimates the error by two or three orders of magnitude. Thus the error bound returned by Algorithm L is loose, albeit safe.

6.2 Componentwise Error Estimate

Algorithm W does not compute a componentwise error bound, nor was it designed to make E_{comp} small, so only its true error E_{comp} is analyzed in this section.

The most important observation is that Algorithm 2 delivers a tiny componentwise error (E_{comp} strongly converged) and a slightly larger error bound (B_{comp} also strongly converged) as long as $\kappa_{\text{comp}} < {}^{1}/_{\gamma\varepsilon_w}$, i.e. for all componentwise "well"-conditioned matrices (545427 out of 2 million cases). This is the best possible behavior we could expect.

The second important observation is that for harder problems, where $\kappa_{\text{comp}} \geq {}^{1}/_{\gamma\varepsilon_w}$, Algorithm 2 also does well, getting strong convergence in both E_{comp} and B_{comp} in 94% of the cases.

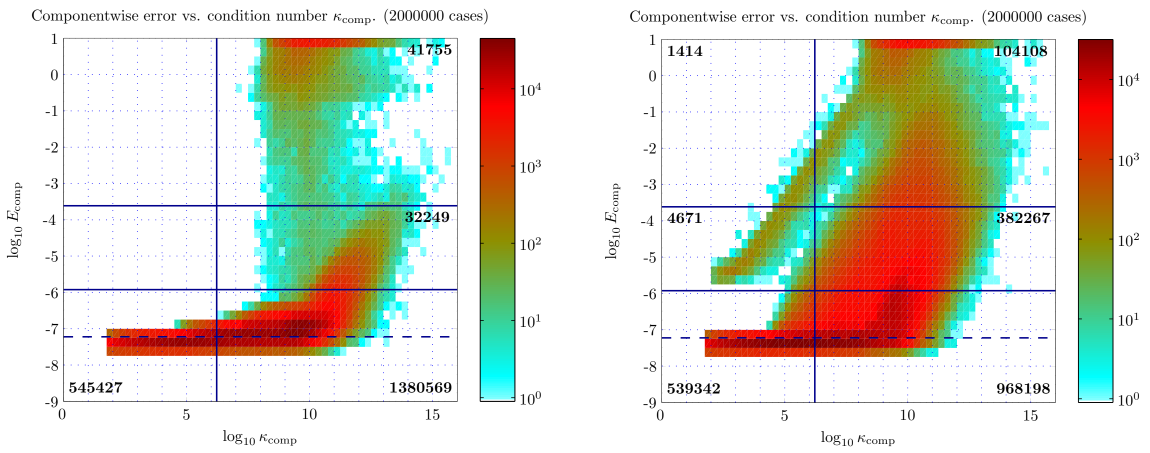


Figure 5: Componentwise error vs. κ_{comp}. Left: Algorithm 2 with $\rho_{\text{thresh}} = 0.5$. Right: Algorithm W.

In the rest of this section, we explore the experimental data in more detail, describing what goes wrong when we fail to get strong convergence.

The two plots in Figure 5 show the 2D histograms of the test problems plotted according to their componentwise error E_{comp} and condition number κ_{comp} for the Algorithms 2 and W. These graphs may be interpreted similarly to those in Figure 2, which were described in the last section.

As already observed, Figure 5 shows that for "well"-conditioned problems Algorithm 2 attains strong convergence of E_{comp} in all cases. Algorithm W, which was not designed to get small componentwise errors, does slightly worse, with strong convergence in 99% of the cases (539342 out of 545427), and weak or no convergence in the other 1%, including a few truly well-conditioned problems.

The second important conclusion is that for harder problems, with $\kappa_{\text{comp}} \geq {}^{1}/_{\gamma\varepsilon_w}$ (1454573 cases) Algorithm 2 still does very well, exhibiting strong convergence of E_{comp} in 95% of cases (1380569 out of 1454573). Algorithm W does much worse, exhibiting strong convergence of E_{comp} in only 66.6% of cases (968198 out of 1454573), and failing to converge at all more than twice as frequently (104108 versus 41755).

In contrast, with Algorithm L the error grows roughly proportional to the condition number, (see [7, Fig. 9c]). Strong convergence of E_{comp} is very rare, only 7.3% of "well"-conditioned cases, and not at all for ill-conditioned cases.

As in the last section, we note that a small error E_{comp} is helpful only if the algorithm also produces a comparably small error bound B_{comp}. Consider Figure 6, whose interpretation is the same as that of Figure 3 in the last section.

First consider the left-hand plot in Figure 6.2. All 545427 "well"-conditioned problems appear in the dark line at the bottom left of the plot; with $B_{\text{comp}} = \gamma\varepsilon_w$ and E_{comp} slightly smaller, usually around ε_w. Thus we can trust Algorithm 2 to deliver a tiny error and a slightly larger error bound as long as $\kappa_{\text{comp}} < {}^{1}/_{\gamma\varepsilon_w}$. By subtracting out these 545427 "well"-conditioned cases, we get the distribution of results ($E_{\text{comp}}, B_{\text{comp}}$) for all ill-conditioned cases. Most still yield strong convergence in both quantities (94%).

Finally, the right-hand plot in Figure 6.2 shows the 2D histogram of the ratio ${}^{B_{\text{comp}}}/_{E_{\text{comp}}}$ plotted

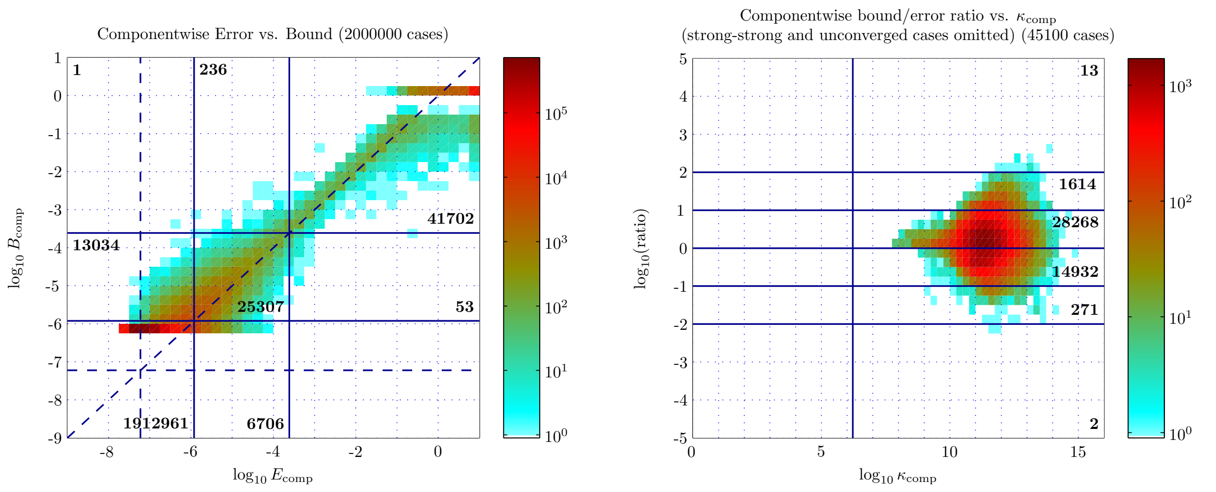


Figure 6: Componentwise error vs. bound for Algorithm 2. Left: B_{comp} vs E_{comp} for all 2M cases. Right: Ratio ($B_{\text{comp}}/E_{\text{comp}}$) vs. κ_{comp}, Algorithm 2 with $\rho_{\text{thresh}} = 0.5$. Cases with strong convergence (in both E_{comp} and B_{comp}) and cases with no convergence ($B_{\text{comp}} > \sqrt{\varepsilon_w}$) are omitted for clarity.

against κ_{comp}. This graph is similar in interpretation to Figure 4. In contrast Figure 4, we see there are more cases where Algorithm 2 attains neither strong convergence in both true error and error bound, nor convergence failure in both: 45100 cases versus 1800 (both out of 2 million, so rather few either way). Considering underestimates of the error, we see that there are also more cases where the ratio is less than 0.1, 273 vs. 7.

6.3 Iteration Counts

Table 1 shows statistics on the number of iterations required by the algorithms; in other words, the number of iterations was not limited ($i_{\text{thresh}} = \infty$), and we allowed the algorithms to run until the other stopping criteria were satisfied. The data is broken down into "well"-conditioned problems versus ill-conditioned problems, and by the number of iterations where the solution x is represented in working (single) precision versus doubled precision. The behavior of Algorithm 2 for different values of ρ_{thresh} is also shown, and will be discussed later.

For "well"-conditioned problems, both Algorithms 2 (for any value of ρ_{thresh}) and W required most 4 iterations and 2.1 iterations on average. Doubled-x iteration was triggered in 55% of cases. On averge 2/3 of the iterations are in single-x and 1/3 in the more expensive doubled-x. Algorithm L is limited to at most 6 iterations in all cases.

For ill-conditioned problems, the median number of iterations used by Algorithm 2 with $\rho_{\text{thresh}} =$ .5 rises to 4, which is still quite modest, but in the worst case we do 33 iterations. Doubled-x iteration is always triggered, right after the first iteration.

6.4 Effects of various parameters in Algorithm 2

Compared to Algorithm W, Algorithm 2 incorporates several new algorithmic ingredients and adjustable parameters. We note that different parameter settings in Algorithm 2 usually do not

	single x			doubled x			total			doubled-x
	max	mean	med	max	mean	med	max	mean	med	incidence
Alg. 2 (any ρ_{thresh})	3	1.5	1	3	0.7	1	4	2.1	2	55%
Alg. W (Wilkinson)							4	2.1	2	
Alg. L (LAPACK)							4	2.6	3	

(a) "Well"-conditioned ($\kappa_{\text{comp}} \leq {}^1/_{\gamma\varepsilon_w}$)

	single x			doubled x			total			doubled-x
	max	mean	med	max	mean	med	max	mean	med	incidence
Alg. 2 ($\rho_{\text{thresh}} = 0.5$)	1	1	1	32	3.6	3	33	4.6	4	100%
Alg. 2 ($\rho_{\text{thresh}} = 0.8$)	1	1	1	89	4.0	3	90	5.0	4	100%
Alg. 2 ($\rho_{\text{thresh}} = 0.9$)	1	1	1	175	4.1	3	176	5.1	4	100%
Alg. 2 ($\rho_{\text{thresh}} = 0.95$)	1	1	1	330	4.3	3	331	5.3	4	100%
Alg. W (Wilkinson)							29	4.0	3	
Alg. L (LAPACK)							6	2.4	2	

(b) Ill-conditioned ($\kappa_{\text{comp}} > {}^1/_{\gamma\varepsilon_w}$)

Table 1: Statistics (max, mean, and median) on the number of iterations required by each algorithm. Algorithms W and L do not use doubled-x scheme so only totals are given.

make any difference for the well-conditioned problems, since all of them quickly converge strongly. However they can make noticeable differences for the very ill-conditioned problems. In this section, we examine the effect of each individual parameter setting, using these difficult problems.

6.4.1 Effect of doubled-x iteration

For ill-scaled systems, the doubled-x iteration is very useful in order to get accurate results for the small components in the solution. To measure its benefit we ran Algorithm 2 with and without the doubled-x iteration for the two million test cases. See [7, Fig. 13] for 2D histograms of E_{norm} versus B_{norm} and E_{comp} versus *compbnd* with and without doubled-x iteration.

To summarize, with doubled-x iteration, Algorithm 2 obtains 57863 (2.9%) more cases of strong-strong normwise convergence as well as 256030 (12.8%) more cases of strong-strong componentwise convergence. The number of cases where the code reports normwise non-convergence ($B_{\text{norm}} > \sqrt{\varepsilon_w}$) decreases by 179; cases of componentwise non-convergence ($B_{\text{comp}} > \sqrt{\varepsilon_w}$) decreases by 5581. Doubled-x iteration decreases the worst normwise underestimation ratio from 1010 to 230, and the worst componentwise underestimation ratio from 6300 to 320.

6.4.2 Effect of ρ_{thresh}

In Algorithm 2, ρ_{thresh} is used in Criterion 2 to determine when to stop the iteration. A larger ρ_{thresh} allows the algorithm to make progress more slowly and take more steps to converge, which is useful for very ill-conditioned problems. However, a larger ρ_{thresh} may cause more severe overestimates (because of the $(1-\rho_{\text{thresh}})$ factor in the denominator of the error bound) and underestimates (since we are being more aggressive to pursue a small dx).

Table 1 displays the statistics of the total iteration counts for various algorithms. For well-conditioned problems, Algorithm 2 (with various ρ_{thresh}) and Algorithm W all require about the

	Underestimates		Overestimates		No convergence
	$> 100\times$	$> 10\times$	$> 100\times$	$> 10\times$	
Alg. 2 with $\rho_{\text{thresh}} = 0.5$	0	7	1	25	40459
Alg. 2 with $\rho_{\text{thresh}} = 0.8$	0	30	3	151	25452
Alg. 2 with $\rho_{\text{thresh}} = 0.9$	0	34	3	505	22755
Alg. 2 with $\rho_{\text{thresh}} = 0.95$	0	33	14	843	21673
Alg. W (Wilkinson)	6	243	35	2130	42421
Alg. L (LAPACK)	0	0	56494	57262	1942738

(a) Normwise

	Underestimates		Overestimates		No convergence
	$> 100\times$	$> 10\times$	$> 100\times$	$> 10\times$	
Alg. 2 with $\rho_{\text{thresh}} = 0.5$	2	273	13	1627	41939
Alg. 2 with $\rho_{\text{thresh}} = 0.8$	5	463	36	3842	26847
Alg. 2 with $\rho_{\text{thresh}} = 0.9$	6	502	67	7436	24250
Alg. 2 with $\rho_{\text{thresh}} = 0.95$	8	499	140	11094	23297

(b) Componentwise

Table 2: Number of overestimates and underestimates of the error returned by various algorithms. Cases with strong convergence in both true error and error bound are not included in the underestimates and overestimates. The number of cases with no convergence is also listed. The category "$> 10\times$" includes the cases under "$> 100\times$".

same number of steps (maximum of 4 with median of 2). For ill-conditioned problems, Algorithm W requires slightly fewer iterations than Algorithm 2 (at the cost of not converging in some cases). For Algorithm 2 it is clear that a larger ρ_{thresh} may potentially need a much larger number of iterations. However, a large number of iterations is required only when the problem is extremely hard and happens relatively rarely (hence the median stays at 4).

Table 2 gives the number of overestimates and underestimates of the error bounds returned by Algorithm 2, as a function of ρ_{thresh}: The number of unconverged cases drops nearly in half as ρ_{thresh} increases from 0.5 to 0.95, at the cost of more severe overestimates and underestimates.

6.5 "Cautious" versus "aggressive" parameter settings

To summarize, by setting ρ_{thresh} and i_{thresh} smaller or larger, Algorithm 2 can be made "cautious" or "aggressive". For the cautious setting we choose $\rho_{\text{thresh}} = .5$ and $i_{\text{thresh}} = 10$. For the aggressive setting we choose $\rho_{\text{thresh}} = .9$ and $i_{\text{thresh}} = 100$. The cautious parameter setting works reliably on all "well"-conditioned problems and so we use it as the default setting in the algorithm. The cautious setting also works for a large fraction of the most ill-conditioned problems, achieving strong normwise convergence in 96.4% of cases and strong componentwise convergence in 94.0% of cases. Failure to converge is indicated by returning $B_{\text{norm}} = 1$ and/or $B_{\text{comp}} = 1$, meaning no accuracy is guaranteed. We expect that most users would prefer this cautious mode as the default. On the other hand, the aggressive parameter setting could be used for very ill-conditioned problems, at the slightly higher risk of long convergence, or extreme over/underestimates of error.

6.6 Limitations of Refinement and our Bounds

In [7, Sec. 7] we explore selected ill-conditioned examples in detail to see how Algorithm 2 behaves when it severely overestimates or underestimates the true error. Briefly, underestimates occur when the algorithm believes it has converged, but to a slightly wrong answer, and overestimates occur either because of early termination or a true error being "accidentally" much smaller than $\|dx\|$ would indicate.

We also tried certain specially constructed difficult problems, besides the ones described in Section 5. When faced with Rump's outrageously ill-conditioned matrices [19] and random x, our algorithm either successfully solved the systems ($O(\varepsilon_w)$ errors and bounds) or correctly reported failure to converge. We also tried exactly singular matrices with slightly inconsistent right-and-sides, but where LU factorization still succeeded because of roundoff. Algorithm 2 computed error bounds that were fairly close to 1, i.e. exceeded our reliability threshold of $\sqrt{\varepsilon_w}$ by a large margin, and so correcly indicated that the results were unreliable. We also tried problems with exactly zero solution components; when these were a result of A^{-1} and b having appropriate zero structure, these zero components were computed exactly with zero error bounds. Otherwise, the code correctly decided not to return a componentwise error bound.

7 Conclusions and Future Work

We have presented a new variation on the extra precise iterative refinement algorithm for the solution of linear equations. With negligible extra work we can return a bound on the maximum relative error in any solution component, as well as the normwise error bound. We prove this by means of an error analysis exploiting column scaling invariance of the algorithm. With the availability of the extended precision BLAS standard, the algorithm can be implemented portably. Based on a large number of numerical experiments we show that the algorithm converges quickly for all but the worst conditioned problems and that the corresponding error bounds (normwise or componentwise) are *completely* reliable, (i.e. when the corresponding computed condition number is less than $1/\gamma\varepsilon_w$). The algorithm also converges for a large fraction of the extremely ill-conditioned problems although the error bounds occasionally underestimate the true error. Some difficulties with the badly scaled problems (i.e. with greatly varying solution components) can be overcome by using extra precision for the updated x (the so-called double-x iteration).

Systems like MATLAB [17] that solve $Ax = b$ return a warning when A is nearly singular, based on a condition estimator. This estimator, like Algorithm 2, costs very little beyond the triangular factorization for medium to large n. Therefore, these systems could use iterative refinement as a default, issuing a warning only if the system is not guaranteed to be fully accurate, because κ_{norm} (or κ_{comp}) is too large.

Our algorithm applies to all the other LAPACK [1] and ScaLAPACK [4] linear system solvers. Additional structure in symmetric and banded systems may allow better error estimates or earlier termination. Section 2's error analysis needs to be extended to these systems. Algorithm 2 also applies to our parallel sparse direct solver SuperLU_DIST [16], where static pivoting instead of partial pivoting is used for numerical stability.

The majority of computers contain processors with Intel's IA32 architecture [13], and support 80-bit floating-point arithmetic in hardware. Future work will extend Algorithm 2 and its error analysis to use this kind of extended precision.

Extra-precise iterative refinement may also be applied to least squares problems, eigenvalue problems, and any other method of numerical linear algebra. Ultimately, the goal should be to achieve a guaranteed tiny error for *all* algorithms in LAPACK as long as an appropriate condition number is sufficiently less than $1/\varepsilon_w$, and to to this for a small cost beyond the most straightforward solution (eg $O(n^2)$ extra on top of $O(n^3)$). This will not necessarily be possible in all cases (eg small extra cost for narrow band problems) but this level of accuracy should be made available for the standard problems of numerical linear algebra.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide, Release 3.0.* SIAM, Philadelphia, 1999. URL `http://www.netlib.org/lapack/lug/`. 407 pages.

[2] *IEEE Standard for Binary Floating Point Arithmetic.* ANSI/IEEE, New York, Std 754-1985 edition, 1985. URL `http://grouper.ieee.org/groups/754/`.

[3] Åke Björck. Iterative refinement and reliable computing. In M.G. Cox and S.J. Hammarling, editors, *Reliable Numerical Computation*, pages 249–266. Oxford University Press, 1990.

[4] L. S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, 1997. URL `http://www.netlib.org/scalapack/slug/`. 325 pages.

[5] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, Z. Maany, F. Krough, G. Corliss, C. Hu, B. Keafott, W. Walster, and J. Wolff v. Gudenberg. Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard. *Intern. J. High Performance Comput.*, 15(3-4), 2001. URL `http://www.netlib.org/blas/blast-forum/`.

[6] H.J. Bowdler, R.S. Martin, G. Peters, and J.H. Wilkinson. Handbook series linear algebra: Solution of real and complex systems of linear equations. *Numerische Mathematik*, 8:217–234, 1966.

[7] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy. Error bounds from extra precise iterative refinement. Technical Report UCB/CSD-04/1344, Computer Science Division, University of California at Berkeley, 2004. also LAPACK Working Note 165.

[8] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide.* SIAM, Philadelphia, 1979.

[9] N. J. Higham. A survey of condition number estimation for triangular matrices. *SIAM Review*, 29:575–596, 1987.

[10] N. J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Trans. Math. Soft.*, 14(4):381–396, 1988.

[11] N. J. Higham. Experience with a matrix norm estimator. *SIAM J. Sci. Stat. Comput.*, 11: 804–809, 1990.

[12] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, PA, second edition, 2002. ISBN 0-89871-521-0. URL `http://www.ma.man.ac.uk/~higham/asna/`.

[13] *IA-32 Intel^{TM} Architecture Software Developer's Manual, Volume 1: Basic Architecture.* Intel Corporation, 2004. URL `http://developer.intel.com/design/pentium4/manuals/index_new.htm#sdm_vol1`. Order #253665.

[14] Andrzej Kiełbasiński. Iterative refinement for linear systems in variable-precision arithmetic. *BIT*, 21:97–103, 1981.

[15] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, and D. J. Yoo. Design, Implementation and Testing of Extended and Mixed Precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, 2002. URL `http://www.nersc.gov/~xiaoye/XBLAS`.

[16] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, June 2003. URL `http://doi.acm.org/10.1145/779359.779361`.

[17] MathWorks, Inc. Matlab^{TM}. URL `http://www.mathworks.com/`.

[18] Cleve B. Moler. Iterative refinement in floating point. *Journal of the Association for Computing Machinery*, 14(2):316–321, 1967. URL `http://doi.acm.org/10.1145/321386.321394`.

[19] Siegfried M. Rump. A class of arbitrarily ill conditioned floating-point matrices. *SIAM Journal on Matrix Analysis and Applications*, 12(4):645–653, October 1991. URL `http://locus.siam.org/SIMAX/volume-12/art_0612049.html`.

[20] S.M. Rump. Solving algebraic problems with high accuracy. In U.W. Kulisch and W.L. Miranker, editors, *A New Approach to Scientific Computation*, pages 51–120. Academic Press, 1983.

[21] S.M. Rump. Verified computation of the solution of large sparse linear systems. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 75:S439–S442, 1995.

[22] J. N. Snuyder. On the improvement of the solutions to a set of simultaneous linear equations using the ILLIAC. *J. Math. Tables other Aids Comput.*, 9(52):177–184, 1955.

[23] G. W. Stewart. *Introduction to Matrix Computations.* Academic Press, New York, 1973. ISBN 0-89871-355-2. xiii+441 pp.

[24] V. Strassen. Gaussian Elimination is not optimal. *Numerical Mathematica*, 13:354–356, 1969.

[25] L.N. Trefethen and R.S. Schreiber. Average-case stability of gaussian elimination. *SIAM Journal on Matrix Analysis and Applications*, 11(3):335–360, 1990. URL `http://locus.siam.org/SIMAX/volume-11/art_0611023.html`.

[26] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. ISBN 0-486-67999-3. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.